Introduction to Intel-TBB

Cecina Babich Morrow

2024-003-01

Introduction to Intel-TBB

Intel-TBB stands for Intel's Threading Building Blocks. It is now a part of something called oneAPI, which is a set of tools and libraries that Intel has developed to help you leverage the power of multi-core processors. Intel-TBB is a C++ template library for parallel programming.

OpenMP vs. Intel-TBB

Both are third-party tools that are not part of C++ itself. OpenMP is a set of compiler directives that you can use to tell the compiler to parallelize your code. Intel-TBB is a C++ template library that you can use to parallelize your code.

Intel-TBB uses standard C++ templates and classes, so it is more portable than OpenMP.

Using Intel-TBB

The following code compiles a C++ program using Intel-TBB.

```
g++ -O3 --std=c++14 test.cpp -Iinclude -ltbb -o test
```

Components of the command:

- `-O3` tells the compiler to optimize the code
- `--std=c++14` tells the compiler to use the C++14 standard (you can use other more recent versions if you want)
- `-Iinclude` tells the compiler to look for header files in the `include` directory
- `-ltbb` tells the compiler to link the Intel-TBB library

Functional programming

Intel-TBB is based on functional programming, which is a programming paradigm that treats functions as objects. Functional programming works easily in R and Python since you can use functions as arguments of other functions natively. Things are a bit more complicated with C++ since C++ relies on variable types. The type of a variable tells you want you can do with that variable and what happens when you act on it. The code referring to a function is a type in C++ just like `int` or `double`, but the syntax gets complicated. For example, the type of a function that takes two `int` arguments and returns an `int` is `int (*)(int, int)`. Fortunately, we can use `auto` instead to avoid needing to write out the type of a function in that way.

Functions as arguments

A template in C++ is a way to write a function or a class without specifying the type of the arguments or the class members. This is useful when you want to write a function or a class that can work with different types of arguments or class members. For example:

```cpp
template<class FUNC, class ARG1, class ARG2>
auto call_function(FUNC func, ARG1 arg1, ARG2 arg2)
{
    auto result = func(arg1, arg2);
    return result;
}
```

In this example, `FUNC` is the type of the function, and `ARG1` and `ARG2` are the types of the arguments. The `auto` keyword tells the compiler to figure out the type of `result` based on the type of `func` and the types of `arg1` and `arg2`.

This means that you can pass function names to other functions. By changing the arguments to a function, you can completely change the function's behavior.

Mapping functions

What if we don't want to write a new function for printing vectors for every type of vector? We can use a template to write a function that works with any type of vector.

We want to move away from loops, which specify a given order for processing the code. By switching to mapping instead, we are allowing flexibility in the order, which will enable parallel processing.

Reductions

A reduction takes a vector as an input and returns a single value, e.g. the sum of all the elements.

Output / Input	1	N
1	function	reduce
N	generator	map